Binary Heaps

A binary heap is another data structure. It implements a priority queue.

Priority Queue has the following operations:

isEmpty
add (with priority)
remove (highest priority)
peek (at highest priority)

What data structures have we seen that has an O(log n) worst-case runtime to add? O(log n) often suggests a balanced binary tree (not necessarily a search tree).

If we can peek at the highest priority in O(1) runtime in the worst case, where must be the highest priority item? At the root.

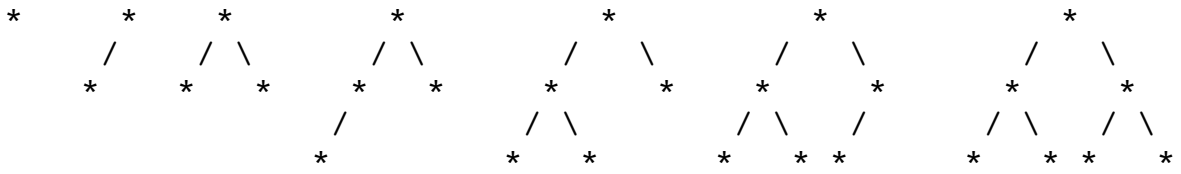
Where would you expect to find the 2nd highest priority item? A child of the root.

Does it matter in which subtree, left or right, that the 2nd highest item is? No. The only requirement is that it should be the highest priority item in its subtree.

Binary heaps have two properties:

1. Shape property: A heap is a complete binary tree – all levels are fully filled, except possibly the bottom level, which may be partially filled from left to right.

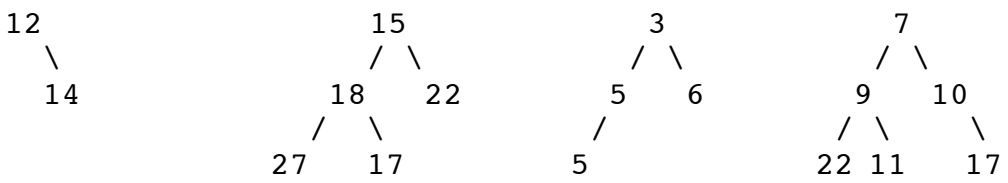
The 7 smallest heap shapes:



2. Order property: parent <= children, if highest priority is the minimum (min-heap)
parent >= children, if highest priority is the maximum (max-heap)

We'll consider min-heaps. Max-heaps are similar.

Which are min-heaps?

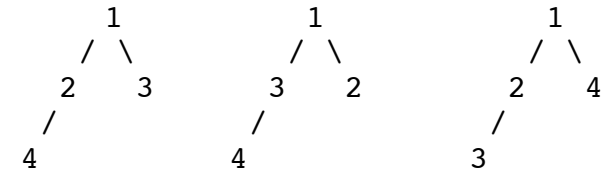


Wrong shape 18-17 out of order Yes! Wrong shape

What are all possible min-heaps on elements 1, 2, 3, 4? How many min-heaps are there?

What shape can the tree have? Only 1 shape.
What value(s) can the root have? 1

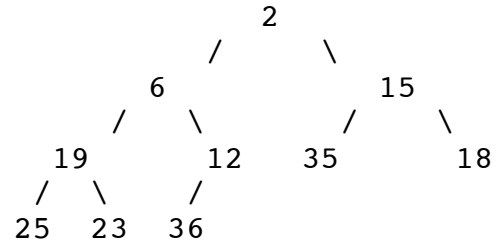
Can 4 be a child of the root? Yes.



Exercise: How many min-heaps are there with elements 1, 2, 3, 4, 5?

For heap with n elements, where can we find the 2nd smallest element? At level 1
Will the 3rd smallest always be at level 1? No.

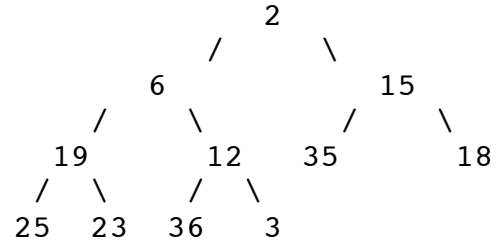
Add an element to a heap:



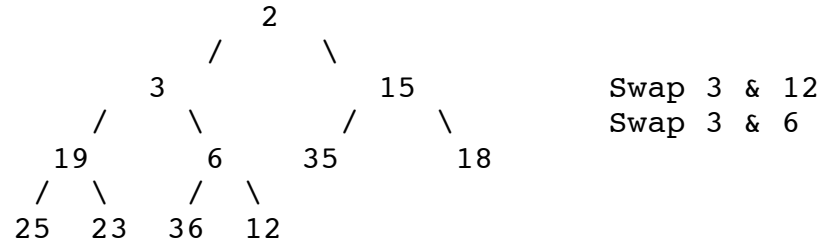
To add 3 to the above heap:

1. Maintain shape property first.
2. Then restore order property.

Step 1: Shape property: *Where must the new element go to keep the tree complete?*
Add 3 as next leaf (child of 12). Ignore that it might violate the order property.



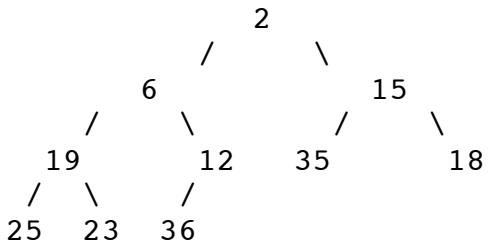
Step 2: Order property: *To where must we move the new element?*
"Heapify" 3 up, by swapping with its parent until the parent is less than or equal to it.



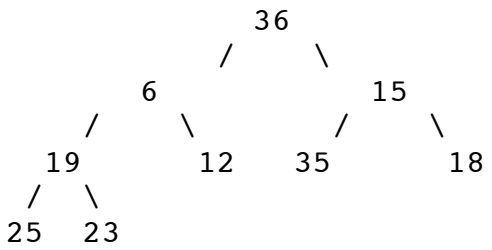
What is the worst-case runtime for add?

Problem: *How can I find where to put the new element?*
Problem: *How do we find the parent of a child?*

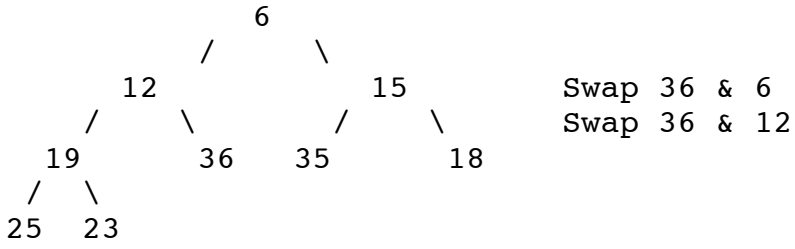
Remove minimum element from the heap:



Step 1: Maintain the shape property: *What element should we use to replace the root we just removed?* Take bottom rightmost element (36) and put it at the root.

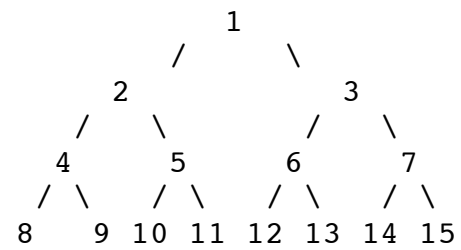


Step 2: Restore the order property second.
Where must we move the new root? "Heapify" new root down until it is less than its parent.
Which node should be the new root? The smaller child.



What is the worst-case runtime for remove?
Problem: *How can I find the element to move to the root?*

Suppose we number the nodes of the binary heap as follows. *Do you see a relationship between a node and its children? A node and its parent?*



For a node numbered *i*

the left child is numbered 2**i*
the right child is numbered 2*i + 1
the parent is numbered *i*/2. (integer division)

Using this indexing we can store a tree in an array (starting at index 1).

Exercise: What would be the index of the left and right child and parent of node numbered i, if the numbering started at 0 instead of 1.

For example, for the binary heap above, after removing 2, the heap would be store in the array as follows:

0	1	2	3	4	5	6	7	8	9
	6	12	15	19	36	35	18	25	23

Notice that the elements are placed in the array one level after the next.

Where do I find the place to put the next leaf? Easy, put at the end of the array.

Why didn't we use arrays to implement a binary tree? There would be gaps in the array whenever there was no corresponding node, and for unbalanced trees the gaps could be huge compared to the number of nodes. But binary heaps are complete trees and the array is filled fully.

Example: Build binary heap with 9 12 8 5 15 2

Add 9

0	1
	9

Add 12

0	1	2
	9	12

Add 8

0	1	2	3
	9	12	8

Swap 9 & 8

0	1	2	3
	8	12	9

Add 5

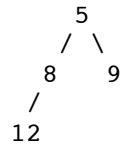
0	1	2	3	4
	8	12	9	5

Swap 12 & 5

0	1	2	3	4
	8	5	9	12

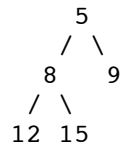
Swap 8 & 5

0	1	2	3	4
	5	8	9	12



Add 15

0	1	2	3	4	5
	5	8	9	12	15



Add 2

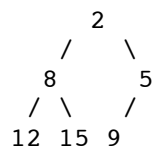
0	1	2	3	4	5	6
	5	8	9	12	15	2

Swap 9 & 2

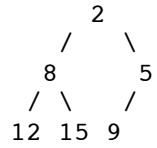
0	1	2	3	4	5	6
	5	8	2	12	15	9

Swap 5 & 2

0	1	2	3	4	5	6
	2	8	5	12	15	9

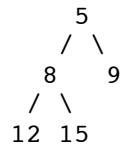


Example: Repeatedly remove the minimum until empty.



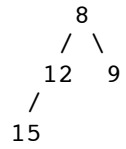
Remove 2

0	1	2	3	4	5	6	
	2	8	5	12	15	9	Replace 2 with 9
0	1	2	3	4	5		
	9	8	5	12	15		Swap 9 with 5
0	1	2	3	4	5		
	5	8	9	12	15		



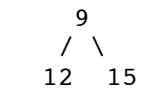
Remove 5

0	1	2	3	4	5		
	5	8	9	12	15		Replace 5 with 15
0	1	2	3	4			
	15	8	9	12			Swap 15 with 8
0	1	2	3	4			
	8	15	9	12			Swap 15 with 12
0	1	2	3	4			
	8	12	9	15			Swap 15 with 12



Remove 8

0	1	2	3	4			
	8	12	9	15			Replace 5 with 15
0	1	2	3				
	15	12	9				Swap 15 with 9
0	1	2	3				
	9	12	15				



And so on.

What is the height of the binary heap? O(log n) – ALWAYS

Runtime:

isEmpty: O(1)

peekMin: O(1)

add:

- best: O(1) – sometime add a large element
- expected: O(log n)
- worst: O(log n)

removeMin: O(log n) – always move a large element to the root

Heap Sort

If remove all the values from a heap, what order will the values be removed? From smallest to largest. We just invented Heapsort

Heap sort Runtime:

1. Build a heap: n* O(log n)
2. Repeatedly remove the minimum: n * O(log n) – always

Total: O(n log n) - best, expected, and worst case

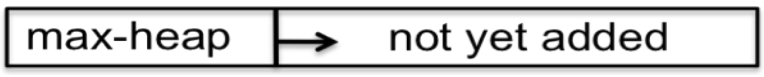
What other sort has the same runtime? merge sort

What is the disadvantage of merge sort? It is not an in-place sort. It needs an auxiliary array.

BUT… We can run heap sort in-place!

Recall how we implement Insertion Sort: We used the front part of the array to hold the sorted data. After each iteration of the outer loop, we added one more element to the sorted part. We will use the same idea to build a heap. This time we will build a max-heap. The reason for a max-heap will become evident.

1. Build a max-heap of size 1, size 2, size 3,... adding each successive element in the array. The first part of the array is the heap and grows. The rest of the array is the source of more data to add.



2. Remove the maximum repeatedly from the heap. Each time we remove an element from the heap, the heap becomes one smaller, and we the use newly available space in the array to put the maximum just removed. That is, we remove the maximum from the heap and put in the last index, then remove next largest from the heap and put it in the second-to-last index, and so on. In particular, swap the last element in the heap with the first, and then heapify down the first element to restore the order property.

   The first part of the array is the max-heap and the rest of the array is the largest data elements in ascending order. Note that we use a max-heap, so that the maximum is at the end of the array.